Sensitivity Analysis in SUNDIALS: Current and Coming Attractions

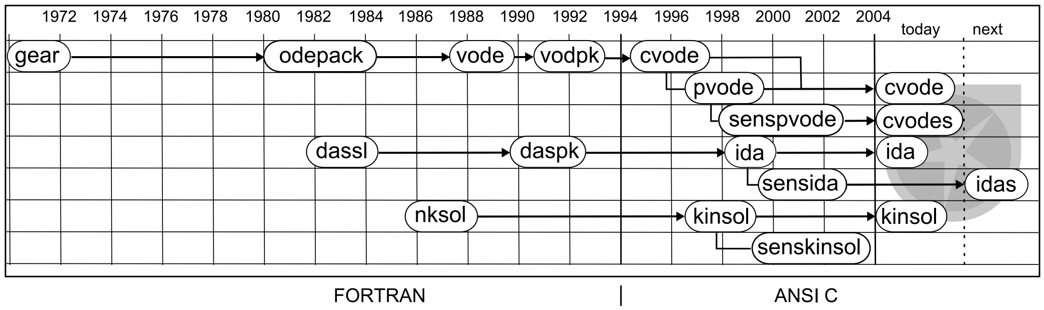
Radu Şerban

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

Eighth DOE ACTS Collection Workshop
August 22, 2007

Outline

Development timeline



Solution of large systems in parallel motivated writing (or rewriting) solvers in C

- **CVODE** C rewrite of **VODE/VODPK** [Cohen,Hindmarsh, 1994]
- **PVODE** parallel extension of **CVODE** [Byrne,Hindmarsh, 1998]
- **KINSOL** C rewrite of **NKSOL** [Taylor,Hindmarsh, 1998]
- **IDA** C rewrite of **DASPK** [Hindmarsh,Taylor, 1999]

Codes organized as a suite (SUNDIALS) in 2002

[Hindmarsh,Brown,Grant,Lee,S.,Shumaker,Woodward, 2005]

Sensitivity capable solvers in SUNDIALS

- **CVODES** [S.,Hindmarsh, 2002]
- **IDAS** [S., 2007]

The SUNDIALS solvers

CVODE - ODE solver

- Variable-order, variable-step BDF (stiff) or implicit Adams (nonstiff)
- Nonlinear systems solved by Newton or functional iteration
- Linear systems solved by direct (dense or band) or iterative solvers

IDA - DAE solver

- Variable-order, variable-step BDF
- Nonlinear system solved by Newton iteration
- Linear systems solved by direct or iterative solvers

KINSOL - nonlinear solver

- Inexact Newton solver (GMRES, BiCG-Stab, or TFQMR iterative linear solvers)
- (Modified) Newton solver (dense or band direct linar solvers)

CVODES

- Sensitivity-capable (forward & adjoint) version of **CVODE**

IDAS

- Sensitivity-capable (forward & adjoint) version of **IDA**

Salient features of the SUNDIALS solvers

CASC

- Philosophy: **Keep codes simple to use**
- Written in C
  - Fortran interfaces: **FCVODE**, **FIDA**, and **FKINSOL**
  - Matlab interfaces: **SUNDIALSTB** (**CVODES**, **IDAS**, and **KINSOL**)
- Written in a **data structure neutral** manner
  - No specific assumptions about data
  - Alternative data representations and operations can be provided
- Modular implementation: vector modules, linear solver modules, preconditioner modules
- Require minimal problem information, but offer user control over most parameters
- Sensitivity Analysis
  - Philosophy: **Require minimal changes to enable SA**
  - Provide both FSA and ASA support

What is SA?

CASC

Definition

Broadly speaking, *sensitivity analysis* (SA) is the study of how the variation in the output of a model (numerical or otherwise) can be apportioned, qualitatively or quantitatively, to different sources of variation.

Local sensitivity analysis (dynamical systems)

$$F(\dot{y}, y, p) = 0$$
$$g(p) = g(y(p), p)$$

where $y \in R^n$ and $g : R^n \times R^{N_p} \longrightarrow R$.
Considering the Taylor expansion of g around the nominal value p

$$g(p + \delta p) = g(p) + \frac{dg}{dp}\delta p + (\delta p)^T \frac{d^2 g}{dp^2}\delta p + O(\|\delta p\|^3)$$

we define

- 1^{st} order SA problem:
  find the gradient $\frac{dg}{dp} = g_y y_p + g_p$
- 2^{nd} order SA problem:
  find the Hessian $\frac{d^2 g}{dp^2} = (I_{N_p} \otimes g_y) y_{pp} + g_{py}^T y_p + y_p^T g_{yp} + y_p^T g_{yy} y_p + g_{pp}$

Why do SA?

- **Model evaluation**
  Finding most and least influential parameters
- **Model reduction**
  Reducing model complexity, while preserving its input-output behavior
- **Data assimilation**
  Merging observed information into a model in order to improve its accuracy
- **Uncertainty quantification**
  Characterizing (quantitatively) and reducing uncertainty in model predictions
- **Dynamically-constrained optimization**
  Improving model response (better performance, better agreement with observations, etc.)

How to perform SA? (1)

Parameter-dependent ODE system

Model: $F(\dot{y}, y, p) = 0$

Output functional: $g(p) = g(y(p), p)$

Finite-difference sensitivity analysis

$$\frac{dg}{dp_i}(p) \approx \frac{g(p + e_i \delta p_i) - g(p)}{\delta p_i}$$

or

$$\frac{dg}{dp_i}(p) \approx \frac{g(p + e_i \delta p_i) - g(p - e_i \delta p_i)}{2\delta p_i}$$

where e_i is the i-th column of the identity matrix and δp is a vector of perturbations.

How to perform SA? (2)

Parameter-dependent ODE system

Model: $F(\dot{y}, y, p) = 0$
Output functional: $g(p) = g(y(p), p)$

Forward sensitivity analysis

$$F_{\dot{y}} \dot{s}_i + F_y s_i + F_{p_i} = 0$$

and

$$dg/dp = g_y s_i + g_{p_i}$$

Cost $\sim (1 + N_p) \times \text{cost(sim)}$

Adjoint sensitivity analysis

$$-(\lambda^T F_{\dot{y}})' + \lambda^T F_y + h(g) = 0$$

and

$$\langle F_p, \lambda \rangle \rightarrow \nabla_p g(p)$$

Cost $\sim (1 + N_g) \times \text{cost(sim)}$

FSA for ODE and DAE systems

- Parameter dependent system: $F(\dot{y}, y, p) = 0\,, \quad y(t_0) = y_0(p)$
- Output functional: $g(y, p)$
- Sensitivity systems: $(i = 1, 2, \ldots, N_p)$

$$F_{\dot{y}} \dot{s}_i + F_y s_i + F_{p_i} = 0\,, \quad s_i(t_0) = y_{0p_i}$$

- Gradient of output functional:

$$\frac{dg}{dp} = g_y S + g_p$$

where $S = [s_1, s_2, \ldots, s_{N_p}]$ is the *sensitivity matrix*.

- Sensitivity equations depend on p but not on g.
- A linear combination Su of all sensitivity vectors can be computed with $\sim$ twice the cost of computing y.

ASA for ODE and DAE systems (1)

Model: $F(\dot{y}, y, p) = 0\,, \quad y(t_0) = y_0(p)$

Output functional: $G(p) = \int_{t_0}^{t_f} g(y,p)\, dt$

Gradient: $\frac{dG}{dp} = \int_{t_0}^{t_f} (g_p - \lambda^T F_p)\, dt - (\lambda^T F_{\dot{y}} y_p)|_{t_0}^{t_f}$

Adjoint system: $(\lambda^T F_{\dot{y}})' - \lambda^T F_y = -g_y\,, \quad \lambda(t_f) = ?$

Consistent final conditions [Cao,Li,Petzold,S., 2003]

index-0 and index-1 DAE

$$F(\dot{y}, y) = 0 \Rightarrow (\lambda^T F_{\dot{y}})' - \lambda^T F_y = -g_y$$

Can use any $\lambda(t_f)$ such that

$$\left(\lambda^T F_{\dot{y}}\right)_{t=t_f} = 0$$

(in particular $\lambda(t_f) = 0$) and therefore

$$\frac{dG}{dp} = \int_{t_0}^{t_f} \left(g_p - \lambda^T F_p\right)\, dt + \left(\lambda^T F_{\dot{y}}\right)_{t=t_0} y_{0p}$$

ASA for ODE and DAE systems (2)

Model: $F(\dot{y}, y, p) = 0\,, \quad y(t_0) = y_0(p)$

Output functional: $G(p) = \int_{t_0}^{t_f} g(y,p)\, dt$

Gradient: $\frac{dG}{dp} = \int_{t_0}^{t_f} (g_p - \lambda^T F_p)\, dt - (\lambda^T F_{\dot{y}} y_p)|_{t_0}^{t_f}$

Adjoint system: $(\lambda^T F_{\dot{y}})' - \lambda^T F_y = -g_y\,, \quad \lambda(t_f) = ?$

Consistent final conditions [Cao,Li,Petzold,S., 2003]

Hessenberg index-2 DAE

$$\begin{aligned} \dot{y}^d &= f^d(y^d, y^a, p) \\ 0 &= f^a(y^d, p) \end{aligned} \quad \Rightarrow \quad \begin{aligned} \dot{\lambda}^d &= -A^T \lambda^d - C^T \lambda^a - g_{y^d}^T \\ 0 &= -B^T \lambda^d - g_{y^a}^T \end{aligned}$$

Search for final conditions of the form $\lambda^d(t_f) = (C^T \xi)_{t=t_f}$

$$t = t_f \Rightarrow \left\{ \begin{array}{l} \lambda^{dT} B = -g_{y^a} \Rightarrow \xi^T CB = -g_{y^a} \Rightarrow \xi^T = -g_{y^a}(CB)^{-1} \\ f^a(y^d, p) = 0 \rightarrow C y_p^d = -f_p^a \Rightarrow \lambda^{dT} y_p^d = -yi^T f_p^a \end{array} \right.$$

$$\Rightarrow \lambda^{dT}(t_f) = -\left(g_{y^a}(CB)^{-1} C\right)_{t=t_f}$$

$$\Rightarrow \frac{dG}{dp} = \int_{t_0}^{t_f} \left(g_p + \lambda^{dT} f_p^d + \lambda^{aT} f_p^a\right)\, dt + \lambda^{dT}(t_0) y_{0p}^d + \left(g_{y^a}(CB)^{-1} f_p^a\right)_{t=t_f}$$

Sensitivity of $g(y(t_f), t_f, p)$ (1)

Use: $\frac{dg}{dp}(t_f) \equiv \frac{d}{dt_f}\frac{dG}{dp}$

Gradient: $\frac{dg}{dp}(t_f) = \left(g_p - \lambda^T F_p\right)_{t=t_f} + \int_{t_0}^{t_f} \mu^T F_p \, dt - \left(\mu^T F_{\dot y} y_p\right)_{t=t_0} - \frac{d(\lambda^T F_{\dot y} y_p)}{dt_f}$

Adjoint system: $(\mu^T F_{\dot y})' - \mu^T F_y = 0\,, \quad \mu(t_f) = ?$

Consistent final conditions [Cao,Li,Petzold,S., 2003]

implicit ODE

$$F(\dot y, y) = 0 \Rightarrow (\mu^T F_{\dot y})' - \mu^T F_y = 0$$

At $t = t_f$

$$\lambda^T F_{\dot y} = 0 \Rightarrow (\lambda^T F_{\dot y})' + \mu^T F_{\dot y} = 0$$

and therefore

$$\mu^T(t_f) = \left(F_{\dot y}^{-1} g_y\right)_{t=t_f}$$

If $y_0 = y_0(p) \Rightarrow$

$$\frac{dg}{dp}(t_f) = g_p(t_f) + \mu^T(t_0) A(t_0) y_{0p}$$

SUNDIALS | Sensitivity Analysis | Implementation | CVODES and IDAS | Final remarks

Sensitivity of $g(y(t_f), t_f, p)$ (2)

Use: $\frac{dg}{dp}(t_f) \equiv \frac{d}{dt_f}\frac{dG}{dp}$

Gradient: $\frac{dg}{dp}(t_f) = \left(g_p - \lambda^T F_p\right)_{t=t_f} + \int_{t_0}^{t_f} \mu^T F_p \, dt - \left(\mu^T F_{\dot y} y_p\right)_{t=t_0} - \frac{d(\lambda^T F_{\dot y} y_p)}{dt_f}$

Adjoint system: $(\mu^T F_{\dot y})' - \mu^T F_y = 0\,, \quad \mu(t_f) = ?$

Consistent final conditions [Cao,Li,Petzold,S., 2003]

Hessenberg index-1 DAE

$$\begin{aligned} \dot y^d &= f^d(y^d, y^a) \\ 0 &= f^a(y^d, y^a) \end{aligned} \quad \Rightarrow \quad \begin{aligned} \dot\mu^d &= -A^T \mu^d - C^T \mu^a \\ 0 &= B^T \mu^d + D^T \mu^a \end{aligned}$$

$A = \partial f^d / \partial y^d$, $B = \partial f^d / \partial y^a$, $C = \partial f^a / \partial y^d$, $D = \partial f^a / \partial y^a$ nonsingular

$$\mu^{dT}(t_f) = \left(g_{y^d} - g_{y^a} D^{-1} C\right)_{t=t_f}$$

If $y_0^d = y_0^d(p) \Rightarrow$

$$\frac{dg}{dp}(t_f) = g_p(t_f) + \mu^{dT}(t_0) y_{0p}^d$$

Sensitivity of $g(y(t_f), t_f, p)$ (3)

Use: $\frac{dg}{dp}(t_f) \equiv \frac{d}{dt_f}\frac{dG}{dp}$

Gradient: $\frac{dg}{dp}(t_f) = \left(g_p - \lambda^T F_p\right)_{t=t_f} + \int_{t_0}^{t_f} \mu^T F_p \, dt - \left(\mu^T F_{\dot y} y_p\right)_{t=t_0} - \frac{d(\lambda^T F_{\dot y} y_p)}{dt_f}$

Adjoint system: $(\mu^T F_{\dot y})' - \mu^T F_y = 0\,, \quad \mu(t_f) = ?$

Consistent final conditions [Cao,Li,Petzold,S., 2003]

Hessenberg index-2 DAE

$$\begin{aligned} \dot y^d &= f^d(y^d, y^a) \\ 0 &= f^a(y^d) \end{aligned} \quad \Rightarrow \quad \begin{aligned} \dot\mu^d &= -A^T\mu^d - C^T\mu^a \\ 0 &= B^T\mu^d \end{aligned}$$

$A = \partial f^d/\partial y^d$, $B = \partial f^d/\partial y^a$, $C = \partial f^a/\partial y^d$, CB nonsingular

$$\mu^{dT}(t_f) = \left(g_{y^d} - g_{y^a}(CB)^{-1}\left(CA + \frac{dC}{dt}\right)\right)_{t=t_f}\left(I - B(CB)^{-1}C\right)_{t=t_f}$$

If $y_0^d = y_0^d(p) \Rightarrow$

$$\frac{dg}{dp}(t_f) = g_p(t_f) + \mu^{dT}(t_0) y_{0p}^d$$

Higher-order sensitivities with FSA

- Straightforward extension of 1^{st} order SA.
- Except when dealing with very few parameters, the computational cost is exorbitant.

Example

ODE $\dot y = f(t, y)\,; \quad y(t_0) = y_0(p)\,; \qquad y \in R^n,\, p \in R^{N_p}$

Functional $g(y(p))$

Hessian $$\frac{d^2 g}{dp^2} = \left(I_{N_p} \otimes g_y\right) y_{pp} + y_p^T g_{yy} y_p$$

where

$$\dot y_p = f_y y_p\,; \quad y_p(t_0) = \frac{dy_0}{dp}$$

$$\dot y_{pp} = f_y y_{pp} + \left(I_n \otimes y_p^T\right) f_{yy} y_p\,; \quad y_{pp}(t_0) = \frac{d^2 y_0}{dp^2}$$

Note that $dim(y_p) = nN_p$ and $dim(y_{pp}) = n^2 N_p$.

Higher-order sensitivities with ASA

- Use the same trick as for the “pointwise functional” case: take an additional formal derivatives of the gradient of either G or g.
- The cost of computing a full Hessian is roughly equivalent to the cost of computing the gradient with FSA. However, Hessian-vector products can be cheaply computed with one additional adjoint solve [Ozyurt and Barton, 2005]

Example

ODE $\dot{y} = f(t,y)\,; \quad y(t_0) = y_0(p)\,; \qquad y \in R^n,\, p \in R^{N_p}$

Functional $G(p) = \int_{t_0}^{t_f} g(t,y)\, dt$

Hessian-vector product $\frac{\partial^2 G}{\partial p^2} u = \left[\left(\lambda^T \otimes I_{N_p} \right) y_{pp} u + y_p^T \mu \right]_{t=t_0}$

where

$$- \dot{\mu} = f_y^T \mu + \left(\lambda^T \otimes I_n \right) f_{yy} s\,; \quad \mu(t_f) = 0$$

$$- \dot{\lambda} = f_y^T \lambda + g_y^T\,; \quad \lambda(t_f) = 0$$

$$\dot{s} = f_y s + f_p u\,; \quad s(t_0) = y_{0p} u$$

Generation of the sensitivity equations

Forward sensitivity analysis

- Analytical
- Automatic differentiation (ADIC, ADOLC)
- Directional derivative approximations

$$\begin{cases} f_y s_i \approx \frac{f(t,y+\sigma_y s_i,p) - f(t,y-\sigma_y s_i,p)}{2\sigma_y} \\ f_{p_i} \approx \frac{f(t,y,p+\sigma_i e_i p) - f(t,y,p-\sigma_i e_i)}{2\sigma_i} \end{cases} \qquad \begin{cases} \sigma_i = |\bar{p}_i| \sqrt{\max(rtol, \epsilon)} \\ \sigma_y = \frac{1}{\max(1/\sigma_i, \|s_i\|_{WRMS}/|\bar{p}_i|)} \end{cases}$$

or

$$f_y s_i + f_{p_i} \approx \frac{f(t, y + \sigma s_i, p + \sigma e_i p) - f(t, y - \sigma s_i, p - \sigma e_i)}{2\sigma}$$

where $\sigma = \min(\sigma_i, \sigma_y)$

Adjoint sensitivity analysis

- Analytical
- Reverse automatic differentiation (ADOLC)
- No finite-difference option (cost $\sim$ cost$_{FSA}$)

FSA with implicit solvers

FSA effectively implies solving an extended system of dimension $N \times N_p \Rightarrow$ efficient implementation of an implicit integrator must take advantage of the structure of the sensitivity equations and the fact that they are linearizations of the original DE.

Solutions (for implicit ODE/DAE integrators)

- **Staggered Direct** [Caracotsios and Stewart, 1985]:
  iterate to convergence the nonlinear state system and then solve the linear sensitivity systems
  requires formation and storage of J; errors in $J \longrightarrow$ errors in s
- **Simultaneous Corrector** [Maly and Petzold, 1997]:
  solve simultaneously a nonlinear system for both states and sensitivity variables
  requires formation of sensitivity r.h.s. at every iteration
- **Staggered Corrector** [Feehery, Tolsma, and Barton, 1997]:
  iterate to convergence the nonlinear state system and then use a Newton method to solve for the sensitivity variables
  with iterative linear solvers $\longrightarrow$ effectively Staggered Direct

ASA for nonlinear problems

For nonlinear problems, the forward states are needed in the backward integration phase. Moreover, when using an adaptive integrator, the number of integration steps is not known apriori and the forward and backward DE are evaluated at different times. $\Rightarrow$ need predictable and compact storage of state variables for the solution of the adjoint system and an efficient interpolation scheme.

Solution: checkpointing

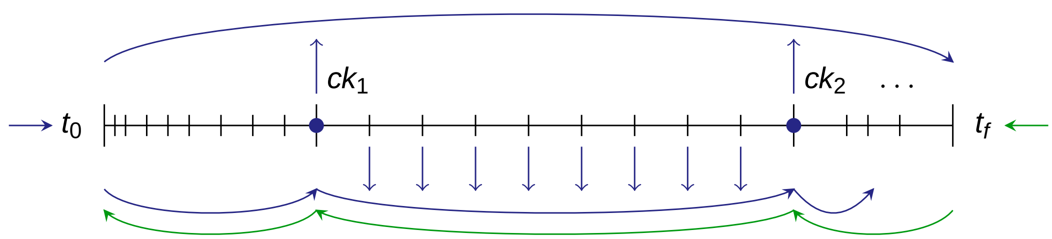
Represents a compromise between efficiency and memory requirements:

- Simulations are reproducible from each checkpoint
- Force Jacobian evaluation at checkpoints to avoid storing it
- Store solution (and possibly first derivative) at all intermediate steps between two consecutive checkpoints
- Interpolation options: cubic Hermite, variable-order polynomial

Checkpointing

Implementation

1. integrate forward step by step
2. dump checkpoint data after a given number of steps
3. continue until t_f.
4. evaluate final conditions for adjoint problem
5. store interpolation data on second forward pass
6. propagate adjoint variables backward in time
7. 1 forward + 1 backward $\leq$ total cost $<$ 2 forward + 1 backward



Discrete vs. continuous sensitivity

Discretization of the adjoint vs. adjoint of the discretization?

- Discrete adjoint of a RK method of order p is an order p discretization of the adjoint equations [Hager, 2000].
- BDF adjoints with variable step are not consistent with the continuous adjoint equation [Sandu, 2003].

Time-dependent PDE solved with MOL: additional issues

- DA: discretization of the adjoint PDE
  - no general derivation method (hard for systems of PDEs).
  - some objective functionals may be *inadmissible* [Arian and Salas, 1997; Giles and Pierce, 1997].
- AD: adjoint of the semi-discretization to ODE/DAE
  - do not require explicit BC for adjoint variables.
  - any cost functional is admissible.
  - may be inconsistent with the adjoint PDE (e.g. when using nonlinear discretization schemes).
  - may not be consistent with *any* PDE close to boundaries [Li and Petzold, 2004].

Salient features of the SUNDIALS SA solvers

- Solution of nonlinear systems for FSA
  - simultaneous corrector
  - staggered corrector
  - modified staggered corrector (**CVODES** only)
- Two-pass checkpointing for ASA
- Two different interpolation modules for ASA:
  - piece-wise cubic Hermite
  - variable-order polynomial
- Support for integration of pure quadrature equations (for the evaluation of integrals in ASA)
- Support for simultaneous integration of multiple adjoint problems and of adjoint problems depending on forward sensitivities (for 2nd order ASA)
- Calculation of consistent initial conditions for state, sensitivity, and adjoint variables (**IDAS** only)
- Generation of the FSA sensitivity systems through directional derivatives

IVP integration with CVODES

Main function

```
/* Set tolerances, initial time, etc.  */
y = N_VNew_Serial(N);
/* Load I.C. into y */
mem = CVodeCreate(CV_BDF, CV_NEWTON);
flag = CVodeSetUserData(mem, my_data);
/* Set other optional inputs */
flag = CVodeInit(mem, rhs, t0, y);
flag = CVodeSStolerances(mem, rtol, atol);
flag = CVDense(mem, N);
for(i=1; i<= NOUT; i++) {
   flag = CVode(mem, tout, y, &t, CV_NORMAL);
   /* Process solution y */
}
N_VDestroy(y);
CVodeFree(&mem);
```

User-supplied functions

required
```
int rhs(realtype t, N_Vector y, N_Vector f, void *data);
```
optional
Jacobian information, preconditioner, rootfinding, quadrature, etc.

FSA with CVODES

Main function (instrumented for FSA)

```
y = N_VNew_Serial(N);
mem = CVodeCreate(CV_BDF, CV_NEWTON);
flag = CVodeSet***(mem,...);
flag = CVodeInit(mem, rhs, t0, y);
flag = CVodeSStolerances(mem, rtol, atol);
flag = CVDense(mem, N);
yS = N_VCloneVectorArray(Ns, N);
flag = CVodeSensInit(mem, Ns, CV_STAGGERED, srhs, yS);
flag = CVodeSensEEtolerances(mem);
flag = CVodeSetSens***(mem,...);
for(i=1; i<= NOUT; i++) {
   flag = CVode(mem, tout, y, &t, CV_NORMAL);
   flag = CVodeGetSens(mem, &t, yS);
}
N_VDestroy(y);
N_VDestroyVectorArray(Ns, yS);
CVodeFree(&mem);
```

User-supplied functions

recommended

```
int srhs(realtype t, N_Vector y, N_Vector f, N_Vector yS,
        N_Vector fS, void *data, N_Vector wrk1, N_Vector wrk2);
```


ASA with CVODES

Main function (instrumented for ASA)

```
y = N_VNew_Serial(N);
mem = CVodeCreate(CV_BDF, CV_NEWTON);
flag = CVodeSet***(mem, my_data);
flag = CVodeInit(mem, rhs, t0, y);
flag = CVodeSStolerances(mem, rtol, atol);
flag = CVDense(mem, N);
flag = CVodeAdjInit(mem, nsteps, CV_POLYNOMIAL);
for(i=1; i<= NOUT; i++) {
   /*flag = CVode(mem, tout, y, &t, CV_NORMAL);
   flag = CVodeF(mem, tout, y, &t, CV_NORMAL, &nckeck);
}
yB = N_VNew_Serial(NB);
flag = CVodeCreateB(mem, CV_BDF, CV_NEWTON, &idxB);
flag = CVodeInitB(mem, idxB, rhsB, tf, yB);
flag = CVodeSet***B(mem, idxB,...);
flag = CVodeB(mem, t0, CV_NORMAL);
flag = CVodeGetB(mem, idxB, &t, yB);
N_VDestroy(y);
N_VDestroy(yB);
CVodeFree(&mem);
```

User-supplied functions

required

```
int rhsB(realtype t, N_Vector y, N_Vector yB, N_Vector fB, void *data);
```

optional

Jacobian information, preconditioner, quadrature, etc.

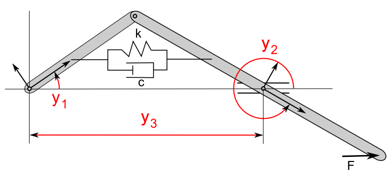
FSA example: slider-crank

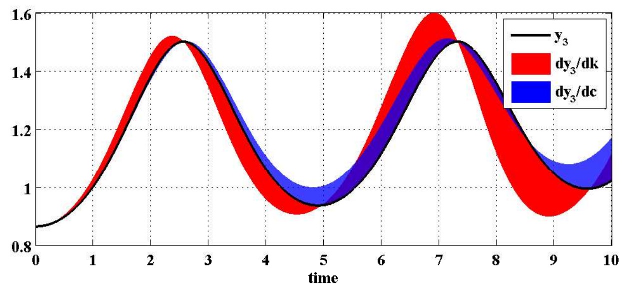
Dynamics stabilized index-2 DAE (GGL formulation)

$$\left\{\begin{array}{rcl} \dot{q} & = & v - \Phi_q^T(q)\mu \\ M(q)\dot{v} & = & Q(q,v,k,c) - \Phi_q^T(q)\lambda \\ \Phi(q) & = & 0 \\ \Phi_q(q)v & = & 0 \end{array}\right.$$

position constraints
velocity constraints

I.C. $q(0) = q_0\,,\quad v(0) = v_0$





ASA example: brusselator

Dynamics Two-species 2D time-dependent PDE

$$\left\{\begin{array}{rcl} u_t & = & \epsilon\Delta u + u^2 v - (B+1)u + A \\ v_t & = & \epsilon\Delta v - u^2 v + Bu \end{array}\right. \qquad \text{in } \Omega = [0,L]^2$$

B.C. $(\partial u/\partial n)|_{\partial\Omega} = (\partial v/\partial n)|_{\partial\Omega} = 0$

I.C. $u(x,y,t_0) = u_0(x,y) \equiv 1.0 - 0.5\cos(\pi y/L)$

$v(x,y,t_0) = v_0(x,y) \equiv 3.5 - 2.5\cos(\pi x/L)$

Output $g(t) = (1/|\Omega|)\int_\Omega u(x,y,t)\,d\Omega$

Adjoint PDE

$$\left\{\begin{array}{rcl} \lambda_t & = & -\epsilon\Delta u - (2uv - B - 1)\lambda + (2uv - B)\mu \\ \mu_t & = & -\epsilon\Delta v - u^2\lambda - u^2\mu \end{array}\right.$$

B.C. $(\partial\lambda/\partial n)|_{\partial\Omega} = (\partial\mu/\partial n)|_{\partial\Omega} = 0$

I.C. $\lambda(x,y,t_f) = 1.0$

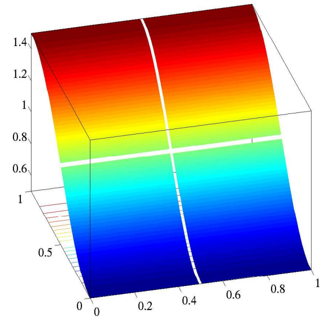
$\mu(x,y,t_f) = 0.0$

Sensitivity $\delta u_0\,,\delta v_0 \Rightarrow \delta g(t_f)$

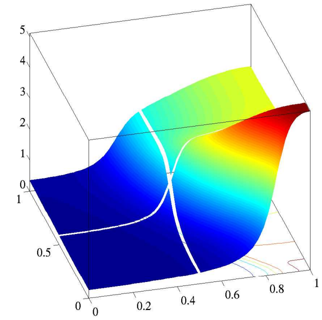
$g(t_f) = (1/|\Omega|)\int_\Omega [\lambda(0,x,y)\delta u_0(x,y) + \mu(0,x,y)\delta v_0(x,y)]\;d\Omega$

ASA example: brusselator

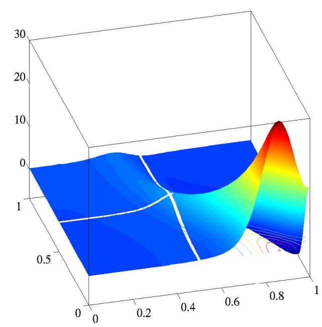
Initial conditions $u(t_0,x,y) \equiv u0(x,y)$



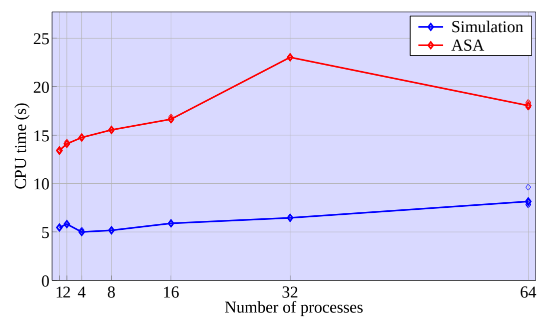
Final solution $u(t_f,x,y)$



Adjoint variables $dg(t_f)/du_0 \equiv \lambda(t_0,x,y)$



Weak parallel scaling
(jacquard.nersc.gov)



Coming attractions in SUNDIALS

SUNDIALS

- **IDAS**- DAE solver with SA capabilities
- **CPODES**- Coordinate projection solver for ODE with invariants
- New linear solver modules
  - Direct dense and band Blas+Lapack linear solvers
  - Support for sparse direct linear solvers

FSA

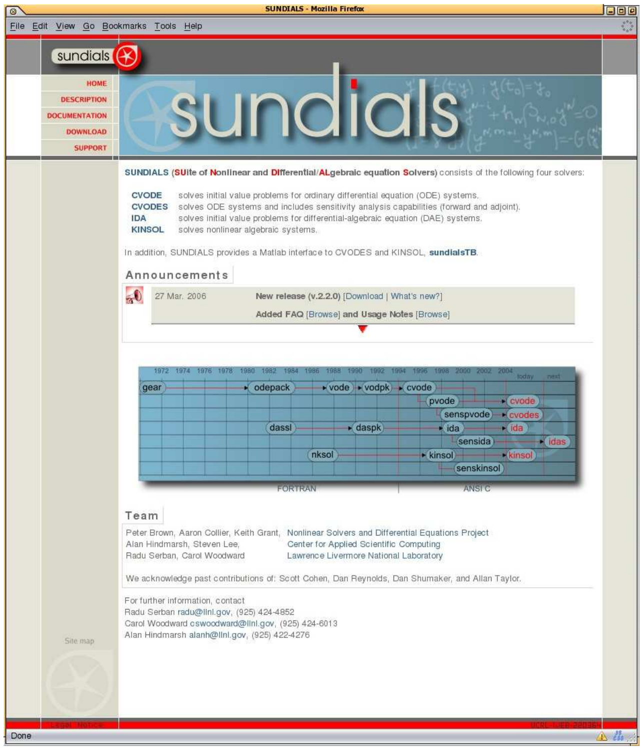
- Support for FSA of pure quadrature variables

ASA

- Support for simultaneous integration of multiple backward problems
- Support for simultaneous FSA-ASA (for 2nd order SA using forward over adjoint)

■ - available in next **SUNDIALS** release (end '07)

www.llnl.gov/casc/sundials



The SUNDIALS suite

- Open source, BSD license
- Extended documentation
- User support

Sensitivity Analysis is...

- useful
  in several different areas (e.g. dynamically-constrained optimmization, UQ).
- enabling
  for various types of analysis (ROM evaluation, global error analysis, solution of certain classes of problems).
- only a few function calls away
  and general-purpose SA solvers are available.
- worthy
  and the effort required to add sensitivity capabilities to a simulation tool is well invested.

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.